# State Management in React

React Workshop - P3

# Events

```
// HTML
<button onclick="onClickHandler()">Click me!</button>

// JSX
<button onClick={onClickHandler}>Click me!</button>
```

- Handling events with React elements is very similar to handling events on DOM elements
- React provides synthetic events to every DOM element events
- In JSX, DOM event handling and React event handling would look alike, but there are some syntax differences
  - React events are named using camelCase, rather than lowercase
  - With JSX you pass a function as the event handler, rather than a string
  - You cannot return false to prevent default behaviour in React. You must call 'preventDefault' explicitly

# State

- 'Props are read-only'
- State – The data which has the ability to change (to be specific, re-render) React components
- To add state to our function component, we can use 'useState' hook

# Hooks

- A hook contains reusable code logic that is separate from the component tree
- They allow us to hook up functionality to our components
- React ships with several built in hooks that we can use out of the box – useState, useContext
- You can define custom hooks and even compose hooks inside custom hooks

# 'useState' Hook

```
const [stateVariable, setStateFn] = useState("defaultValue");
```

- What does calling useState do ?
  - It declares a state variable
  - useState is a new way to use the exact same capabilities that this.state provides in a class
- What do we pass to useState as an argument ?
  - The only argument to the useState() Hook is the initial state
  - Unlike with classes, the state doesn't have to be an object
- What does useState return ?
  - It returns a pair of values: the current state and a function that updates it
- When data within the hook changes, they have the power to re-render the component

# React State – The old way

- Before v16.8.0, the only way to add state to a component was to use a class component
- This required not only a lot of syntax & Does not promotes reusability
- As of today, this code still works. But it will be deprecated in the future
- This will be one of the assignment.

# Forms

- All of the HTML form elements that are available to the DOM are also available as React elements
- Controlled Components
  - In HTML, form elements maintain their own state and update it based on user input
  - In React, mutable state is typically kept in the state property of components, and only updated with setState()
  - We can combine the two by making the React state be the “single source of truth”
  - An input form element whose value is controlled by React is called a 'controlled component'
  - UseRef - 'Uncontrolled Components' (Optional Assigment)

# React Context

- State should be maintained at one place.
- Passing state down the component tree through (prop drilling) is tedious and bug prone
- Context provides a way to pass data through the component tree without having to pass props down manually at every level
- React comes with a function called 'createContext' that we can use to create a new context object

```
const AppContext = createContext();
```

- This object contains two components: a 'Context Provider' and a 'Context Consumer'

# Context Provider

```
<AppContext.Provider value={{ data }}>
  <App />
</AppContext.Provider>
```

- You can place data in React context by creating a context provider
- A context provider is a React component that you can wrap around your entire component tree, or specific sections of your component tree
- We add data to context by setting the 'value' property of the Provider
- This data will be made available to any context consumers found in components wrapped by the Provider component
- It is ok to use multiple context providers for application modularity

# Context Consumer

```
const { data } = useContext(AppContext);
```

- The context consumer is the React component that retrieves the data from context
- 'useContext' Hook
  - The useContext hook obtains values that we need from the context Consumer

# Custom Context Providers

```
const ContextProvider = ({ children }) => {
  const [data, setData] = useState(data);
  return (
    <ContextProvider.Provider value={{ colors, setColors }}>
      {children}
    </ContextProvider.Provider>
  );
};
```

- The context provider can place an object into context, but it can't mutate the values in context on its own
- It needs some help from a parent component, the trick is to create a stateful component that renders a context provider
- When the state of the stateful component changes it will re-render the context provider with new context data
- Any of the context providers children will also be re-rendered with the new context data

# Assignments

1. Compilation of workshop examples.
2. Re-write the contact management example in '02-event-handling' using Class Component.
3. Re-write the example in '04-context' without using 'useContext' Hook (using Context.Consumer component).
4. ~~Understand React-Router and write a basic routing example. Ref: https://reacttraining.com/react-router/web/guides/quick-start~~